ELSEVIER

Discrete Applied Mathematics 52 (1994) 223–232

DISCRETE APPLIED MATHEMATICS

Analysis of scheduling problems with typed task systems

Klaus Jansen

Fachbereich IV, Mathematik und Informatik, Universität Trier, Postfach 3825, D-54286 Trier, Germany

Received 3 March 1992; revised 21 October 1992

Abstract

In this paper we have analysed a scheduling problem where each task has a type and where only a bounded number of processors of each type is available. We show that the typed scheduling problem remains NP-complete for a disjoint set of chains, two types and one processor of each type. If the deadline is a constant number, this problem, given k processors, remains NP-complete for out-trees, in-trees and a disjoint union of chains. In contrast we give a polynomial algorithm if we have an interval order.

1. The scheduling model

We consider the following scheduling problem which is given by:

(1) a finite set T of tasks,

(2) a partial order $D = (T, A)$ on T,

(3) a type $c(t) \in \{1, \ldots, k\}$ for each task $t \in T$,

(4) numbers $m_1, \ldots, m_k \geq 1$ with m_i processors of type i.

The tasks should be executed at discrete time steps; we assume that the time steps are given by positive integers $\{1, \ldots, e_{\max}\} \subset \mathbb{N}$. Each task t must be executed at one time step, on a machine of type $c(t)$ and under the constraints of the partial order. If $(t, t') \in A$ then task t must be executed before task t'. The scheduling problem is to minimize $e_{\max}$ under the assumption that at most m_i machines of type i are used at each time step.

The first complexity results are given for subproblems. If only one type of machine is used, we get the classical PRECEDENCE CONSTRAINT SCHEDULING problem. This problem is NP-complete for a general number of machines [9], but the computational complexity is not known for a constant number of machines. The problem remains NP-complete if the deadline $e_{\max}$ is a constant [6]. But this problem can be solved in polynomial time for some special partial orders. One class is the set of in-trees, where each vertex has one successor with exception of one vertex the root which has no successor. The reverse trees with predecessors instead of successors are

SSDI 0166-218X(93)E0035-W

called out-trees. For both orders the problem can be solved with Hu's algorithm [4] in polynomial time. Another algorithm is known for interval orders [8] $D = (T, A)$ where each task t can be represented as an interval I_t in the real line with $(t, t') \in A$ iff for each $x \in I_t$, $y \in I_{t'}$ we have $x < y$.

The general problem with different types is more difficult. It was shown in [3] that this problem is NP-complete even if we have an arbitrary order, two types and one processor of each type. This result is dominated by the NP-completeness [1] for a forest. We give a stronger domination of this result given by the NP-completeness for a disjoint union of chains. After that we have analysed the complexity of this problem for constant deadline $e_{\max}$. In contrast we show that the typed scheduling problem can be solved in polynomial time for the interval orders. This generalizes the existence of a polynomial-time algorithm for the interval orders [5] where all numbers m_i are equal to one.

2. Chains and trees

We consider chains of vertices which are vertex and edge disjoint. To show the NP-completeness we give a transformation from the NP-complete problem 3-PARTITION [2] to the typed scheduling problem. The 3-PARTITION problem is formulated as follows.

Instance: Set A with $3m$ elements, a bound $B \in \mathbb{N}$ and a size $\ell(a)$ for each $a \in A$ with $B/4 < \ell(a) < B/2$ and $\sum_{a \in A} \ell(a) = mB$.

Question: Is there a partition of the A into m disjoint sets $A_1, \ldots, A_m$ such that for each $1 \leqslant i \leqslant m$, $\sum_{a \in A_i} \ell(a) = B$?

We note that for a solution of 3-PARTITION each set A_i must have exactly three elements of A; otherwise the set A_i cannot have the total size B. Since the proof has also been used for a scheduling problem with additional resources [7], we give only an idea of the proof.

Theorem 2.1. *The typed scheduling problem remains* NP*-complete, if we have a disjoint union of chains, two types of tasks and one processor of each type.*

Proof. By transformation from 3-PARTITION. Given a set A, sizes $\ell(a)$, a bound B and a number m we construct several chains:

(1) For each $a_i \in A$ we construct a chain of $\ell(a_i)$ vertices $b_i^{(j)}$ of type 1 and with $\ell(a_i)$ vertices $d_i^{(j)}$ of type 2 at the end:

$$b_i^{(1)} \to \cdots \to b_i^{(\ell(a_i))} \to d_i^{(1)} \to \cdots \to d_i^{(\ell(a_i))}.$$

(2) Additionally we take one chain which consists of m subchains of length B with vertices $c_i^{(j)}$ of type 2 for $1 \leqslant j \leqslant B$ and m subchains of length B with vertices $e_i^{(j)}$ of type 1 for $1 \leqslant j \leqslant B$ in an alternating form:

$$c_1^{(1)} \to \cdots \to c_1^{(B)} \to e_1^{(1)} \to \cdots \to e_1^{(B)} \to$$
$$\cdots \to c_m^{(1)} \to \cdots \to c_m^{(B)} \to e_m^{(1)} \to \cdots \to e_m^{(B)}.$$

Then we get the equivalence that there is a 3-partition of A in m sets each with size B iff there is a scheduling with two processors one of type 1 and one of type 2 and length $2mB$. □

Corollary 2.2. *The typed scheduling problem remains* NP*-complete, if we have an in-tree or an out-tree, two types of tasks and one processor of each type.*

If the deadline $e_{\max}$ is a constant equal to three, it was shown [6] that the classical untyped scheduling problem is NP-complete for an arbitrary partial order and m processors. For $e_{\max} = 2$ the classical untyped scheduling problem can be solved in polynomial time.

Theorem 2.3. *The typed scheduling problem can be solved in linear time* $O(|T|)$ *for* $e_{\max} = 2$, *an arbitrary partial order* $D = (T, A)$ *and* m_i *processors of type* $1 \leqslant i \leqslant k$ *where* k *is the number of different types.*

Proof. Consider that for $e_{\max} = 2$ there cannot be a path $a \to b \to c$ in the order of length three. If a task t has at least one successor, he must be executed at step one on a processor of type $c(t)$. If a task t has at least one predecessor he must be executed at the second step. After the assignment of these tasks to processors $1, \ldots, k$ it remains $m_i^{(1)}$ processors for the first step and $m_i^{(2)}$ processors for the second step. The computation of these numbers can be done in linear time $O(|T|)$. Define T_i as the tasks of type i without predecessors and successors. An assignment to the remaining processors is only possible if $m_i^{(1)} + m_i^{(2)} \geqslant |T_i|$ for each type $1 \leqslant i \leqslant k$. The computation of the sizes $|T_i|$ and the test of the inequalities can also be done in linear time. □

An important question is the computational complexity for special orders like an in-tree, an out-tree or a forest of out-trees and in-trees. For the classification of these problems we use the NP-complete problem 3-SAT (Satisfiability) [2] which is formulated as follows:

Instance: Set X of variables x_j for $1 \leqslant j \leqslant n$ and a collection of sets $B_1, \ldots, B_m$ such that each $B_i = \{y_{i,1}, y_{i,2}, y_{i,3}\}$ with $y_{i,\ell} = x_j$ or $y_{i,\ell} = \overline{x_j}$.

Question: Is there a map $\psi : X \to \{0, 1\}$ with $\psi(\overline{x_j}) = 1 - \psi(x_j)$ such that in each set B_i at least one element is assigned to one?

Theorem 2.4. *The typed scheduling problem remains* NP*-complete for a forest of out-trees and constant deadline* $e_{\max} = 3$.

Proof. By transformation from 3-SAT to the typed scheduling problem. Let $B_1, \ldots, B_m$ be an instance of 3-SAT with clauses $B_i = \{y_{i,1}, y_{i,2}, y_{i,3}\}$. For such an instance of 3-SAT we give an instance of the typed scheduling problem as follows.

- Let $T = \{z_{i,j} \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant 3\} \cup \{x_j, \overline{x_j} \mid 1 \leqslant j \leqslant n\} \cup \{a_i, b_i, d_i \mid 1 \leqslant i \leqslant m\}$.
- For the partial order take the following edges $\{(x_j, z_{i,k}) \mid x_j = y_{i,k}\}$ and $\{(\overline{x_j}, z_{i,k}) \mid \overline{x_j} = y_{i,k}\}$ or each $1 \leqslant j \leqslant n$ and $\{(a_i, b_i), (b_i, d_i)\}$ for each $1 \leqslant i \leqslant m$.
- The types of the tasks are given by $c(x_j) = c(\overline{x_j}) = j$, $c(z_{i,k}) = c(d_i) = n + i$ and $c(a_i) = c(b_i) = n + m + i$.
- Choose one machine of type $i \in \{1, \ldots, n, n + m + 1, \ldots, n + 2m\}$ and three machines of type $i \in \{n + 1, \ldots, n + m\}$.

Now we show the equivalence that α is satisfiable iff there is a schedule of length three.

Let $\psi : X \to \{0, 1\}$ be an assignment to the variables where in each set B_i at least one element gets the value one. For that we give a schedule $S : T \to \{1, \ldots, 3\}$. For the literals we define $S(x_j) = 2 - \psi(x_j)$ and $S(\overline{x_j}) = 2 - \psi(\overline{x_j})$. Therefore the literals with value one are executed at the first step and the others at the second step. For the chains of length three we define $S(a_i) = 1$, $S(b_i) = 2$ and $S(d_i) = 3$. Since $c(d_i) = n + i$ there can be executed only two further tasks from each set B_i.

Let $j_1, \ldots, j_m \in \{1, \ldots, 3\}$ be a possible choice of indices such that the corresponding literals y_{i,j_i} are one. We can define $S(z_{i,j_i}) = 2$, because the corresponding literals are executed one step before. After that we can execute the other tasks at the third step, because for each set B_i there are only two remaining tasks.

For the other direction let $S : T \to \{1, \ldots, 3\}$ be a feasible schedule. The tasks in the chains $a_i \to b_i \to d_i$ must be executed step by step. Therefore there can be executed only two further tasks of type $n + i$ at the third step. Without loss of generality we can assume that exactly one of the tasks $x_j, \overline{x_j}$ is executed at the first step and the other at the second step. The other tasks $z_{i,j}$ can only be executed after step one. But only the tasks $z_{i,j}$ with $S(y_{i,j}) = 1$ are possible at the second step for each $1 \leqslant i \leqslant m$. To have a feasible schedule with length three one task z_{i,j_i}, must be executed at the second step for each $1 \leqslant i \leqslant m$. Define $\psi(x_j) = 2 - S(x_j)$ and $\psi(\overline{x_j}) = 1 - \psi(x_j)$. Then we get that $\psi(y_{i,j_i}) = 1$ and therefore the element $y_{i,j_i} \in B_i$ has the value one. □

Theorem 2.5. *The typed scheduling problem remains* NP*-complete for an out-tree and constant deadline* $e_{\max} = 4$.

Proof. By transformation from the scheduling problem for a forest of out-trees by adding a new task a, connecting task a with each root $r \in T$ in the forest by an edge (a, r) and by increasing $e_{\max}$ by one. Then only the task a can be executed at step one and after this step we have the original forest. □

We note that the typed scheduling problem can be solved in linear time for out-trees and $e_{\max} = 3$. For that use the fact that the root must be scheduled alone at the first step and apply the general approach for $e_{\max} = 2$.

Reversing all the precedences of the partial order we can prove similar results for a forest of in-trees. The reversal of an out-tree is an in-tree and the legal schedules for the original instance are exactly the reversals of the legal schedules for the reverse instance.

Theorem 2.6. *The typed scheduling problem remains* NP*-complete for a forest of in-trees and constant deadline* $e_{\max} = 3$.

Theorem 2.7. *The typed scheduling problem remains* NP*-complete for an in-tree and constant deadline* $e_{\max} = 4$.

Similar to the out-trees the typed scheduling problem can be solved for in-trees and $e_{\max} = 3$ in linear time. We can prove a similar result also for a disjoint union of chains, but with a greater constant $e_{\max} = 4$ and a more complicated proof. For the proof of NP-completeness we use the problem SAT with the property that each variable and its complement occurs twice in the clauses B_i. The NP-completeness of this restricted SAT problem can be proved from the original SAT problem [2].

Instance: Set $X \cup \bar{X}$ of variables x_j and their complement $\overline{x_j}$ and a collection of sets $B_1, \ldots, B_m \subset X \cup \bar{X}$ such that $|\{i \mid x_j \in B_i\}| = 2$ and $|\{i \mid \overline{x_j} \in B_i\}| = 2$ for $1 \leqslant j \leqslant n$.

Question: Is there a map $\psi : X \to \{0, 1\}$ with $\psi(\overline{x_j}) = 1 - \psi(x_j)$ such that in each set B_i at least one element is assigned to one?

Theorem 2.8. *The typed scheduling problem remains* NP*-complete for a disjoint union of chains and constant deadline* $e_{\max} = 4$.

Proof. Let $a(x_j), b(x_j)$ be the clauses where x_j occurs and let $a(\overline{x_j}), b(\overline{x_j})$ be the clauses where the complement of x_j occurs. For such an instance of SAT we give an instance of the typed scheduling problem as follows.

- Take the following chains:

(1) for each $1 \leqslant j \leqslant n$ take chains $x_j^{(1)} \to (x_j, 1) \to x_j^{(3)}$ and $x_j^{(2)} \to (x_j, 2) \to x_j^{(4)}$;

(2) for each $1 \leqslant j \leqslant n$ take chains $\overline{x_j^{(1)}} \to (\overline{x_j}, 1) \to \overline{x_j^{(4)}}$ and $\overline{x_j^{(2)}} \to (\overline{x_j}, 2) \to \overline{x_j^{(3)}}$;

(3) for each $1 \leqslant i \leqslant m$ take $d_{i,1} \to d_{i,2} \to h_i \to d_{i,4}$.

- Take the following types for the tasks: $c(x_j^{(k)}) = c(\overline{x_j^{(k)}}) = (k-1)n + j$, $c((y, 1)) = 4n + a(y)$, $c((y, 2)) = 4n + b(y)$ for each literal $y \in \{x_j, \overline{x_j}\}$, $c(h_i) = 4n + i$ and $c(d_{i,j}) = 4n + m + 1$.

- Take one processor of type $1, \ldots, 4n$, then $|B_i|$ processors of type $4n + i \in \{4n + 1, \ldots, 4n + m\}$ and m processors of type $4n + m + 1$.

Then the instance of SAT is satisfiable iff there is a schedule of length four.

Let $\psi : X \cup \bar{X} \to \{0,1\}$ be a map with $\psi(\overline{x_j}) = 1 - \psi(x_j)$ which assigns at least one element in each set B_i the value one. Define as schedule $S(d_{i,j}) = j$, $S(h_i) = 3$. If $\psi(x_j) = 1$ then schedule both the chains which begin with $x_j^{(1)}$ and $x_j^{(2)}$ in the first three steps. For $\psi(\overline{x_j}) = 1$ schedule the chains which begin with $\overline{x_j^{(1)}}$ and $\overline{x_j^{(2)}}$ in step one, two and three. The other chains are assigned to the last three steps. Since at least one element (y_{i,j_i}, k) with $y_{i,j_i} \in B_i$ is assigned to the second step, we have at most $|B_i| - 1$ of the corresponding tasks in step three. Therefore it is possible to execute the remaining tasks together with h_i in this step.

On the other hand let $S : T \to \{1, \ldots, 4\}$ be a feasible schedule. Then it is clear that the tasks in the chains of length four must be executed one by one. Also for each $k \in \{1,2\}$ exactly one of the tasks $x_j^{(k)}, \overline{x_j^{(k)}}$ must be executed in step one and the other in the second step. The same holds for the pairs $x_j^{(k)}, \overline{x_j^{(k)}}$ for $k \in \{3,4\}$ and step three and four.

Additionally it is not possible to schedule a pair $x_j^{(1)}, \overline{x_j^{(2)}}$ at the first step. In this case we must have the tasks $\overline{x_j^{(1)}}, x_j^{(2)}$ at the second step and therefore the endpoints $\overline{x_j^{(4)}}, x_j^{(4)}$ of the chains are scheduled at the fourth step. Since we have only one processor of type $3n + j$ at each step, this is a contradiction. Using this fact it is also not possible to schedule a pair $x_j^{(2)}, \overline{x_j^{(1)}}$ at step one. Therefore either a pair $x_j^{(1)}, x_j^{(2)}$ or a pair $\overline{x_j^{(1)}}, \overline{x_j^{(2)}}$ is scheduled at the first step and the other pair at the second step. Define $\psi(x_j) = 2 - S(x_j^{(1)})$ and $\psi(\overline{x_j}) = 1 - \psi(x_j)$. Since the tasks h_i are scheduled at the third step on a processor of type $4n + i$, for each set B_i it is only possible to schedule $|B_i| - 1$ tasks of type $4n + i$ at the third step. Therefore at least one task of this type must be scheduled at the second step. But this is only possible for tasks (y, k) with $y \in \{x_j, \overline{x_j}\}$ and $k \in \{1,2\}$ where $\psi(y) = 1$. Therefore at least one task in each set B_i gets the value one by the map ψ. □

Now we consider the case that the deadline $e_{\max} = 3$ and that we have a disjoint set of chains. It is clear that in this case each chain can only have length one, two or three. If we have a chain of length three we must schedule the tasks in this chain one by one, because we have no other possibility. Now we can analyse the chains of length two and can test in polynomial time whether the processor numbers are sufficient.

Lemma 2.9. *Let k be the number of processor types where $m_j^{(\ell)}$ processors of type $1 \leqslant j \leqslant k$ are available at step $\ell \in \{1,2,3\}$. Let $\{(a_i, b_i) \mid 1 \leqslant i \leqslant n\}$ be the set of disjoint chains of length two in a partial order where the type of the tasks are given by $c(a_i), c(b_i) \in \{1, \ldots, k\}$. Then we can test in polynomial time whether a schedule of length three exists where at most $m_j^{(\ell)}$ processors of type j at step ℓ are used.*

Proof. We can transform this test-problem into a flow-problem which can be solved in polynomial time [2]. Let h_j^A and h_j^B be the number of tasks in $A = \{a_i \mid 1 \leqslant i \leqslant n\}$ and in $B = \{b_i \mid 1 \leqslant i \leqslant n\}$ of type $j \in \{1, \ldots, k\}$. Define a digraph with vertices

$V = \{s,t\} \cup \{x_i \mid 1 \leqslant i \leqslant n\} \cup \{y_{j,\ell} \mid 1 \leqslant j \leqslant k, \ell \in \{1,2,3\}\}$ and with edges $e \in E$ which have a lower capacity $k_1(e)$ and an upper capacity $k_2(e)$ of the following form.

(1) Take edges (s, x_i) for each $1 \leqslant i \leqslant n$ with lower capacity one and upper capacity two.

(2) Take edges $(x_i, y_{c(a_i),1})$ and $(x_i, y_{c(b_i),3})$ for each $1 \leqslant i \leqslant n$ with lower capacity zero and upper capacity one.

(3) Take edges $(s, y_{j,2})$ with lower and upper capacity equal $m_j^{(2)}$.

(4) Take edges $(y_{j,1}, y_{j,2})$ with upper capacity $m_j^{(1)}$ and edges $(y_{j,3}, y_{j,2})$ with upper capacity $m_j^{(3)}$. In both cases the lower capacity is zero.

(5) Take edges $(y_{j,2}, t)$ with lower capacity $h_j^A + h_j^B$ and upper capacity ∞.

Observe that a task a_i can only be executed at step one and two and that a task b_i can only be executed at the last two steps. The first edges (s, x_i) reflect that one of both tasks a_i, b_i must be executed at step one or three. If we have a flow of value one through $(x_i, y_{c(a_i),1})$ we have scheduled a_i at step one and if we have a flow through $(x_i, y_{c(b_i),3})$ we have scheduled b_i at step three. The vertices $y_{j,1}$ test whether we have chosen at most $m_j^{(1)}$ tasks of type j at step one. The same test is done by the vertices $y_{j,2}$ for the third step. Now analyse the second step for each type $j \in \{1,\ldots,k\}$. It is only allowed to schedule at most $m_j^{(2)}$ tasks of this type at step two. But we have scheduled $h_j^A + h_j^B$ tasks minus the flow through the edge $(y_{j,1}, y_{j,2})$ and the flow through the edge $(y_{j,3}, y_{j,2})$. Therefore we must have

$$h_j^A + h_j^B - f((y_{j,1}, y_{j,2})) - f((y_{j,3}, y_{j,2})) \leqslant m_j^{(2)}.$$

This inequality can be transformed into

$$f((y_{j,2}, t)) = f((y_{j,1}, y_{j,2})) + f((y_{j,3}, y_{j,2})) + m_j^{(2)} \geqslant h_j^A + h_j^B.$$

Therefore there is a flow $f: E \to \mathbb{N}_0$ in D with $k_1(e) \leqslant f(e) \leqslant k_2(e)$ if and only if a schedule is possible with the given numbers of processors. □

Theorem 2.10. *The typed scheduling problem can be solved in polynomial time for a disjoint union of chains and constant deadline* $e_{\max} = 3$.

Proof. Consider at first the chains of length three and assign the tasks in these chains to the steps one, two and three. After that only m_j^{ℓ} processors of type $j \in \{1,\ldots,k\}$ at step $\ell \in \{1,2,3\}$ are available. Using the assertion from the lemma we can test in polynomial time whether the chains of length two can be assigned to the three steps. If this is possible we must test whether the isolated tasks can be assigned. But this test depends only on the number of tasks of type $1,\ldots,k$ in the instance. □

3. Interval orders

In this section we give a polynomial algorithm for the interval orders $D = (T, A)$. We assume that the intervals on the real are given in the form $I_t = [t^{(1)}, t^{(2)}]$ with

positive integer endpoints. The corresponding interval graph is denoted by $G_I = (T, E_I)$ with $E_I = \{(t, t') \mid I_t \cap I_{t'} \neq \emptyset\}$. Then the following lemma gives a relation between interval orders and the corresponding interval graph.

Lemma 3.1. *Let $D = (T, A)$ be an interval order, let G_I be the corresponding interval graph. Let $c(t)$ be the type of task t and assume that we have m_i machines of type i. Then there is a partition of G into $e_{\max}$ cliques where each clique has at most m_i tasks of type i iff there is a feasible schedule of D which needs $e_{\max}$ time steps and at most m_i processors of type i.*

Proof. Let $C_1, \ldots, C_{e_{\max}}$ be a partition of G into cliques with at most m_i tasks of type i in each clique C_j. For each clique C_j there is at least one point x on the real line with $x \in \bigcap_{t \in C_j} I_t$. We assume that the cliques are ordered according to these points on the real line. Define as schedule $S(t) = j$ if $t \in C_j$ and prove that S gives a feasible schedule. Let $t, t' \in T$ with $(t, t') \in A$. Since the interval I_t lies on the left side of $I_{t'}$ the corresponding cliques C_j with $t \in C_j$ and C_k with $t' \in C_k$ satisfy $j < k$ and therefore we have $S(t) = j < k = S(t')$. The number of vertices of type i at each time step is less than or equal to m_i and the number of steps is $e_{\max}$.

For the other direction let $S: T \to \{1, \ldots, e_{\max}\}$ be a feasible schedule where for each step $1 \leqslant j \leqslant e_{\max}$ and each type $1 \leqslant i \leqslant k$ we have $|\{t \mid S(t) = j, c(t) = i\}| \leqslant m_i$. Define $C_j = \{t \mid S(t) = j\}$. If C_j is not a clique there are vertices $t, t' \in C_j$ with $\{t, t'\} \notin E_I$. This means that the intervals $I_t \cap I_{t'} = \emptyset$. Therefore I_t lies on the left or on the right side of $I_{t'}$. In both cases we have an arc in $D = (T, A)$ and this means that we have not a feasible schedule. Therefore each set C_j is a clique and we get a partition into $e_{\max}$ cliques with at most m_i tasks of type i in each clique C_j. □

Our scheduling algorithm consists of the following steps:

(1) Sort the tasks due to their right endpoints and get a list $L = t_1, \ldots, t_n$, set $k = 1$.

(2) Take the first task t from the list, set $C_k := \{t\}$, $a = t^{(2)}$, $h_i = 0$ for each type $i \neq c(t)$ and $h_{c(t)} = 1$.

(3) Consider next task t' in the list. If $t'^{(1)} > a$ goto (4), otherwise if $h_{c(t')} + 1 \leqslant m_{c(t')}$ take t' from the list, put t' into C_k and set $h_{c(t')} = h_{c(t')} + 1$.

(4) If $h_i = m_i$ for each type $1 \leqslant i \leqslant k$ or if there is no further task in the list goto (5), otherwise goto (3).

(5) If list L is empty stop, otherwise set $k = k + 1$ and goto (2).

In each iteration the first task t with minimum right endpoint $a = t^{(2)}$ is taken from the list. After that we scan through the list from the left to the right and take at most m_i tasks of type i. But we take only a task t' where the corresponding interval $I_{t'}$ intersect with the interval I_t of the first task.

Theorem 3.2. *The algorithm above solves the problem partition into cliques with at most m_i tasks of type i for an interval graph.*

Proof. Let us assume that for some interval graph our algorithm produces a suboptimal solution. For that compare the optimum solution given by task sets S'_j with the solution S_j of the algorithm above where the index j gives the time step of the execution. We consider an example for a suboptimal solution with minimum number of tasks T and with maximum sized-set S'_1. Now consider the following set $A = \{t \in T \mid t^{(1)} \leqslant a = t_1^{(2)}\}$ where t_1 is the first task from the sorted list. If there is a task $a \notin A$ but in S'_1, task t_1 cannot be executed correctly. Therefore we get $S'_1, S_1 \subset A$. If $S'_1 \subset S_1$ we can enlarge S'_1 and get a smaller example for our algorithm by considering $T \setminus S_1$. By using the maximality of S'_1 we can assume that the number of tasks of type i in S_1 and in S'_1 are equal. Now consider a task $t' \in S'_1 \setminus S_1$ with minimum right endpoint. Using that the numbers of each type are equal we can choose a task $t \in S_1 \setminus S'_1$ with minimum right endpoint and the same type. Let $t \in S'_j$. Since $t \in A$ we can schedule t at step 1. Now we try to schedule t' at step j. Assuming that t' cannot be executed at step j, there must be another vertex $t'' \in S'_{j'}$ with $j' \leqslant j$ which must be executed after t' in the precedence order. In this case we have $t'^{(2)} < t''^{(1)}$. Using that t has the smaller right endpoint as t' we get $t^{(2)} \leqslant t'^{(2)} < t''^{(1)}$. This implies that t must be executed before $t''^{(1)}$ and that S is not a feasible schedule. Therefore we can exchange t and t' and get also an optimum schedule. This exchange step can be iterated such that we get an optimum schedule S'' with $S''_1 = S_1$. This is a contradiction to the minimality of $|T|$. □

A naive implementation of the algorithm gives a quadratic time complexity $O(|T|^2)$. But this time complexity can be improved. The time complexity of the sorting step in the algorithm can be bounded by $O(|T|\log(|T|))$. Using the same time we can also compute for each type $1 \leqslant i \leqslant k$ a list of tasks of type i sorted due to the left endpoints. Then, in each iteration and for each type we determine a list of tasks sorted due to the right endpoints where each task starts before the given time step a. In each iteration we update these lists by inserting further starting tasks. Using these sorted lists we can take the tasks of type i starting before time step a. It is clear that this scanning procedure needs $O(|T|\log(|T|))$ time. Using these arguments, we get the following result.

Theorem 3.3. *The typed scheduling problem can be solved in* $O(|T|\log(|T|))$ *time for interval orders.*

References

[1] D. Bernstein, M. Rodeh and I. Gertner, On the complexity of scheduling problems for parallel pipelined machines. IEEE Trans. Comput. 38 (1989) 1308–1313.

[2] M.R. Garey and D.S. Johnson, Computers and Intractability: A guide to the Theory of NP-completeness (Freeman, New York, 1979).

[3] D.K. Goyal, Scheduling processor bound systems. Technical Report CS-76-036, Washington State University, Pullman, WA.

[4] T.C. Hu, Parallel sequencing and assembly line problems, Oper. Res. 9 (1961) 841–848.
[5] H. Kellerer and G. Woeginger, UET-scheduling with constraint processor allocation, Comput. Oper. Res. 19 (1992) 1–8.
[6] J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity of scheduling under precedence constraints, Oper. Res. 26 (1978) 22–35.
[7] E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Sequencing and scheduling: algorithms and complexity, Report BS-R8909, Center for Math. and CS, Amsterdam.
[8] C.H. Papadimitriou and M. Yannakakis, Scheduling interval-ordered tasks, SIAM J. Comput. 8 (1979) 405–409.
[9] J.D. Ullman, NP-complete scheduling problems, J. Comput. System Sci. 10 (1975) 384–393.